FIG. 1

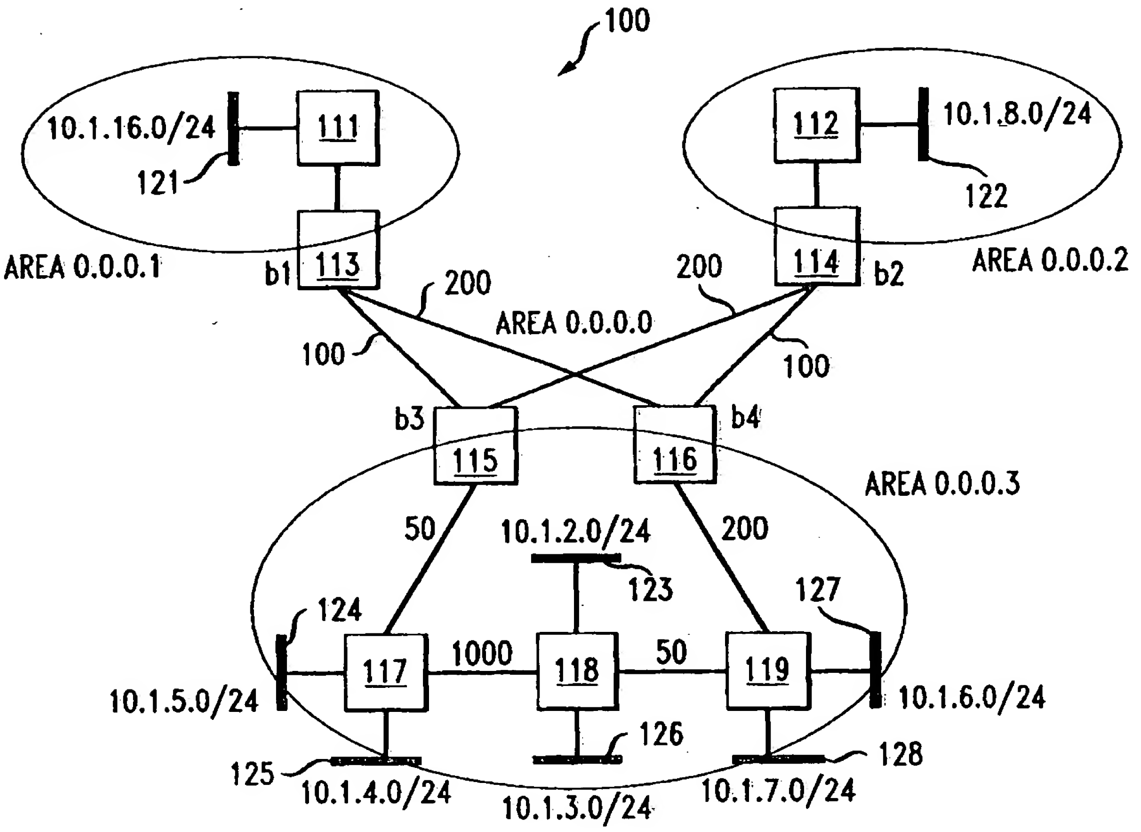

100
10.1.16.0/24
111
121
AREA 0.0.0.1
b1
113
112
10.1.8.0/24
122
114
b2
AREA 0.0.0.2
200
200
AREA 0.0.0.0
100
100
b3
115
b4
116
AREA 0.0.0.3
50
10.1.2.0/24
200
123
127
124
117
1000
118
50
119
10.1.5.0/24
10.1.6.0/24
126
125
10.1.4.0/24
10.1.3.0/24
10.1.7.0/24
128

FIG. 2

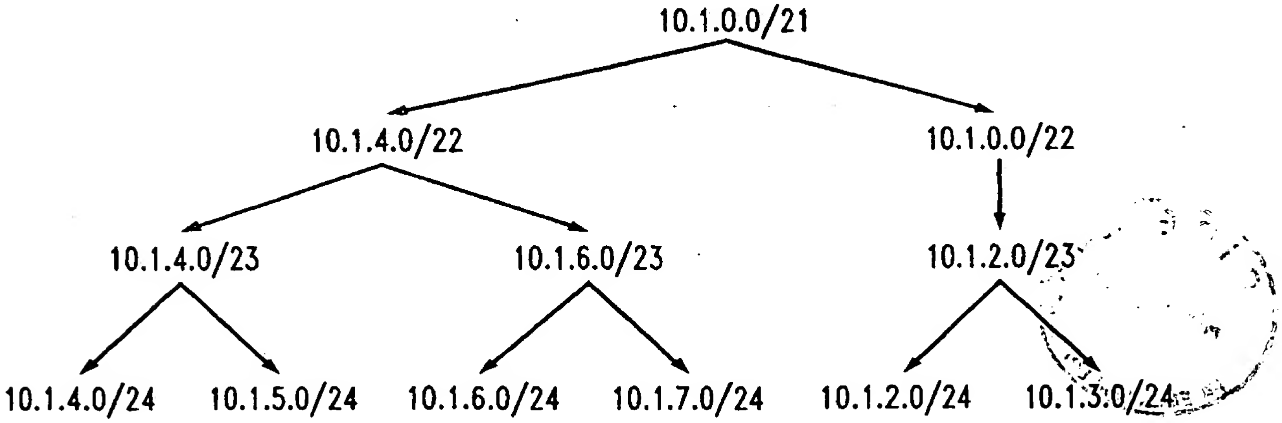

10.1.0.0/21
10.1.4.0/22
10.1.0.0/22
10.1.4.0/23
10.1.6.0/23
10.1.2.0/23
10.1.4.0/24
10.1.5.0/24
10.1.6.0/24
10.1.7.0/24
10.1.2.0/24
10.1.3.0/24

FIG. 3

```
procedure ComputeMinError(Aggregate x, Aggregate y, integer l)
if subTree[x, y, l].computed = true
  return [subTree[x, y, l].error, subTree[x, y, l].aggregates]
  minError := minError1 := minError2 := ∞
if x is a leaf {
  minError1 := $\sum_{s \in S} D(s, t) * (lsp(s, x, \{y\}, W_A) - lsp(s, x))$
  if $l > 0$
    minError2 := $\sum_{s \in S} D(s, t) * (lsp(s, x, \{x\}, W_A) - lsp(s, x))$
  if minError1 ≤ minError2
    [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError1, Ø]
  else
    [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError2, {x}]
}
if x has a single child u {
  [minError1, aggregates1] := ComputeMinError(u, y, l)
  if $l > 0$
    [minError2, aggregates2] := ComputeMinError(u, x, l - 1)
  if minError1 ≤ minError2
    [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError1, aggregates1]
  else
    [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError2, aggregates2 ∪ {x}]
}
if x has children u and v {
  for i := 0 to l {
    [minError1, aggregates1] := ComputeMinError(u, y, i)
    [minError2, aggregates2] := ComputeMinError(v, y, k - i)
    if minError1 + minError2 < minError
      minError := minError1 + minError2
      aggregates := aggregates1 ∪ aggregates2
  }
  for i := 0 to l – 1 {
    [minError1, aggregates1] := ComputeMinError(u, x, i)
    [minError2, aggregates2] := ComputeMinError(v, x, k - i - 1)
    if minError1 + minError2 < minError
      minError := minError1 + minError2
      aggregates := aggregates1 ∪ aggregates2 ∪ {x}
  }
  [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError, aggregates]
}
subTree[x, y, l].computed := true
return [subTree[x, y, l].error, subTree[x, y, l].aggregates]
```

FIG. 4

```
procedure COMBINEMINERROR()
for $i = 1$ to $m$
  for $j = 0$ to $k$ {
    $T_i[j]$.[error, aggregates] := COMPUTEMINERROR(r($T_i$), $\in$, $j$)
    $X_i[j]$.[error, aggregates] := [$\infty$, $\emptyset$]
  }
for $j = 0$ to $k$
  $X_1[j]$.[error, aggregates] := $T_1[j]$.[error, aggregates]
for $i = 1$ to $m$
  for $j = 0$ to $k$
    for $l = 0$ to $j$
      if ($X_{i-1}[l]$.error + $T_i[j - l]$.error < $X_i[j]$.error) {
        $X_i[j]$.error = $X_{i-1}[l]$.error + $T_i[j - l]$.error
        $X_i[j]$.aggregates = $X_{i-1}[l]$.aggregates $\cup$ $T_i[j - l]$.aggregates
      }
```

FIG. 5

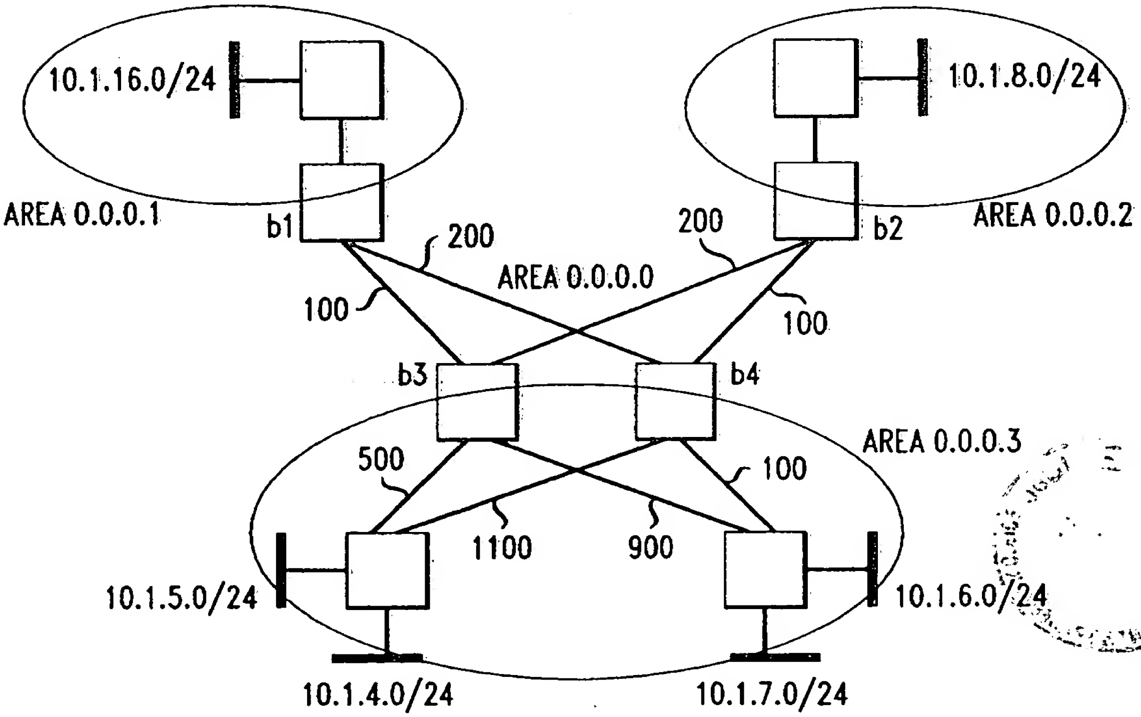


FIG. 6

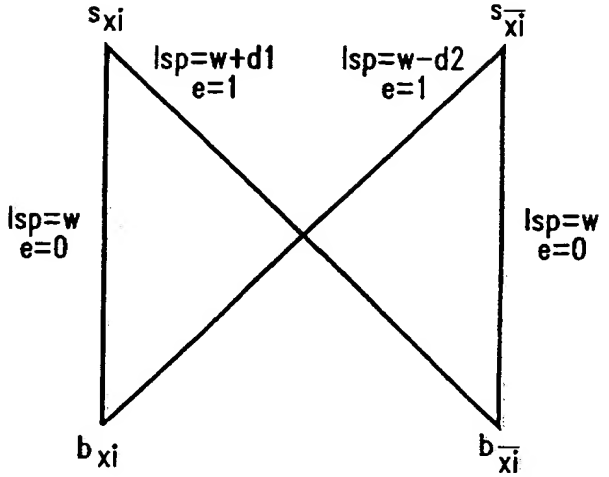
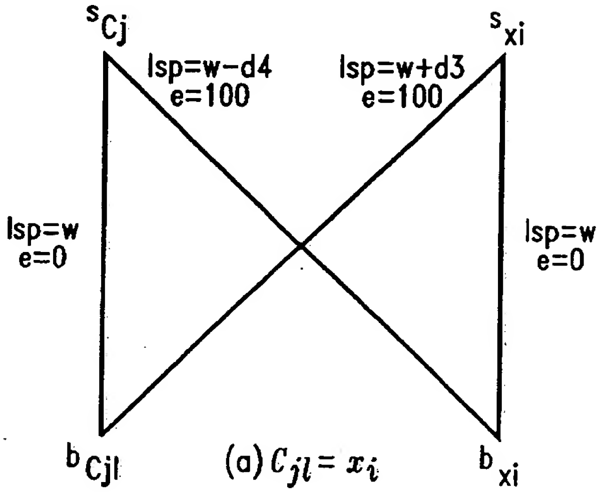
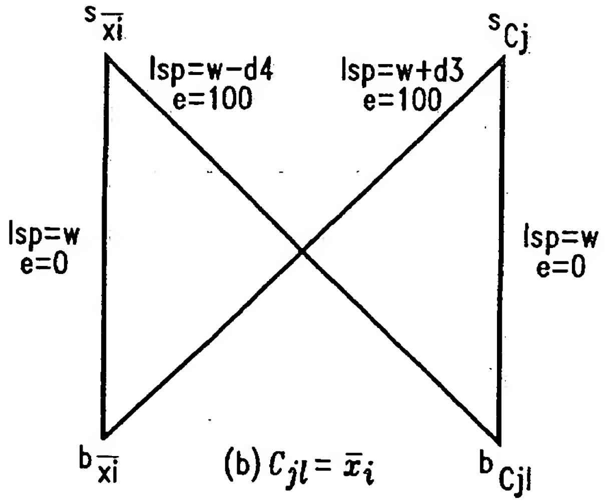


FIG. 7A



(a) $C_{jl} = x_i$

FIG. 7B



(b) $C_{jl} = \bar{x}_i$

FIG. 8

```
procedure COMPUTEWEIGHTSCUMULATIVE ()
  for each $b \in \mathbf{B}_i$ set $W_{min}(b) := 0$
  for $i$ :=1 to $r$ {
    $W := W_{min}$
    Choose a random subset $R \subseteq B_i$ of ABRs
    for each $b \in R$ set $W(b)$ to a random weight in $[0, L]$
    if $\sum_{s \in} se(s, B(s, W)) < \sum_{s \in} se\ (s, B(s, W_{min}))$
      $W_{min} := W$
  }
  return $W_{min}$
```

FIG. 9

```
procedure ComputeWeightsMax(Q)
    1.for each b $\in$ $\boldsymbol{B}_i$ set Wold(b) := 0
    2. while (P$b_2$B

i Wold(b) $\leq$ (

j $\boldsymbol{B}_i$ j*(j $\boldsymbol{B}_i$ j-l)

2 ) *lspmax) f3. Let

Q0 be a new set of inequalities that result when the value Wold(b) is substituted for each variable W (b)only on the LHS of each inequality in Q 4. Set Wnew(b) to the smallest possible value such that each inequality in Q0 is satisfied when Wnew(b) is substituted for variable W (b) in Q0 5. if Wnew= Wold 6. return Wnew 7. else 8. Wold := Wnew 9.g 10. return "there does not exist a weight assignment W"
```

FIG. 10

```
procedure COMPUTEWEIGHTSTWOABR()
  Set $V_{opt} := v(s_1)$, $E := E_{opt} := \sum_{s \in} se(s, b_1)$
  for $j$:=l to $n$ {
    $E := E + e(s_j, b_2) - e(s_j, b_1)$
    if $E < E_{opt}$
      $V_{opt} := v(s_{j+1}), E_{opt} := E$
  }
  return $V_{opt}$
```